Visual Report Generation Tool for Grammar-Based Classifier System

Olgierd Unold, Agnieszka Kaczmarek, and Łukasz Culer

***Abstract*—Grammar-based Classifier System (GCS) is one of the evolutionary approached context-free grammar induction methods. Any learning process creates a large amount of data, which is hard to analyze in a raw form. In the present paper we aim to present a reporting tool, created to facilitate analysis and conclusion drawing by presenting learning data in a neat and readable form, yet fully conveying its complexity.**

***Index Terms*—Context–free grammars, data visualization, grammatical inference.**

I. Introduction

Grammatical inference is an intensively studied area of research that sits at the intersection of several fields including finite state machines, formal grammars, machine learning, language processing and the learnability theory. Many problems of grammar inference are important in theory, and most of them belong to the class NP-hard.

Let $G = (\Sigma,V,P,s)$ be a context-free grammar (CFG), where Σ is a finite set of terminal symbols, $V \cap \Sigma = \emptyset$, V is a finite set of non-terminal symbols, P is a finite set of production rules, $P \subset V \times (\Sigma \cup V)^*$, and $s \in V$ is a start symbol. Production rules of a context-free grammar are usually written in the form: $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (\Sigma \cup V)^*$. Let $L(G)$ denote the set of strings (the string language) generated by G over the alphabet Σ, for which there exists a derivation $s \Rightarrow^* x$ using rules in P, where $x \in \Sigma^*$. The time complexity to recognize whether a certain sentence (string) belongs to a given context-free language $L(G)$ is $O(n^3)$, where n is the length of a sentence.

Due to large complexity of generating the set P of production rules, the induction of a context-free grammar G, regardless of the learning model used (identification in the limit, PAC, MDL, etc.), is computationally complex. The literature implicitly indicates that G is not learnable in polynomial time [1]. One approach to deal with this difficulty is to use methods of machine learning.

II. Grammar-Based Classifier System

Learning Classifier System (LCS) [2], which combines two biological metaphors - evolution and learning - is part of this area. It develops a set of classifiers in a form of condition-action rules adapted to a set of positive and negative input examples.

Grammar-based Classifier System, introduced in 2005 [3], derives from LCS, implementing a new representation of classifiers population, the scheme of classifiers' matching the environmental state and the methods of exploring new classifiers.

Classifiers population in GCS systems has a form of context-free grammar rules set in Chomsky Normal Form (CNF) [4]. Chomsky Normal Form allows only production rules in the form of $A \rightarrow \alpha$ or $A \rightarrow BC$, where A, B, C are non-terminal symbols and α is a terminal symbol. Each classifier is described by the fitness which shows overall classifier adaptation to the environment. There are two constituent elements of fitness:

- *Classic fitness function* - calculated only on the basis of classifier usages in positive examples parsing.
- *Fertility fitness function* – which takes into *consideration* also the position of the classifier in a parsing tree.

The core of the system is CYK (Cocke–Younger–Kasami) parser [5], which enables checking whether the set of rules can parse the input example. CYK array stores a complete history of sentence's parsing, which in turn can be used to give awards to the rules which parse positive examples.

One of the methods used to create new classifiers is a *genetic algorithm*. It is launched with a given probability once the analysis of the train set ends. It can be applied only to non-terminal rules. The following genetic operators are available:

- *Mutation* - the symbol in any position can be changed to another non-terminal symbol with a pre-defined probability.
- *Crossover* - two classifiers exchange their left sides and random symbols from their right sides.
- *Inversion* - a permutation of the symbols located on the right side of the rule.

The algorithm is able to use multiple methods of rule selection, such as a roulette-wheel or a random selection.

The second rule-creating method, *covering*, works during train set analysis and regardless of genetic algorithm. It adds productions that allow continuation of parsing in the current state of the system. There are different types of covering for different types of rules:

- *Start covering* - creates a rule deriving a terminal symbol from the starting symbol *s*.
- *Terminal covering* – a rule producing terminal symbol using any non-terminal symbol, created when the system finds unknown terminal symbol during example parsing.
- *Aggressive covering* - a production rule in the form of $C \rightarrow AB$ is created if symbols A and B can be derived.

Manuscript received July 15, 2017; revised November 10, 2017. This work was supported by National Science Center, grant 2016/21/B/ST6/02158.

The authors are with the Department of Computer Engineering, Wrocław University of Science and Technology, Wrocław, Poland (e-mail: olgierd.unold@pwr.edu.pl, agnieszka.kaczmarek@pwr.edu.pl, lukasz.culer@pwr.edu.pl)

doi: 10.18178/ijmlc.2017.7.6.642

- *Full covering* – aggressive covering modification used to generate a rule with the start symbol on the left side.

For both methods the crowding technique is used in order to maintain diversity in the population.

All the above mentioned elements lead to the induction of final grammar, which shall accept the correct sentences, reject the incorrect ones, and is able to generalize the knowledge in order to classify new sentences correctly.

The GCS systems modification was widely described in the previous works [6], [7].

III. Reporting Tool

As previously mentioned, the goal of this project was to create a tool that presents the collected data in a clear and elegant way. Moreover, generated reports should be easy to share and open with popular software. In order to achieve this goal and design a tool that meets the established assumptions, we decided to implement it using web technologies, like HTML5 and JavaScript. This solution allows our reports to be displayed using a regular web browser.

A. Data flow

Two types of data used for generating report are collected using a plug-in integrated into GCS.

The first type, real-time measurements, are gathered throughout the whole learning process in each iteration. They contain information about each individual that occurred in a given step as well as operations applied to it - starting from age, fitness, usages in proper and invalid parsing, and profit and debt points like in final grammar, to the information if they were removed (with the information about removal reason), added (with the information about its creation process) or retained from the previous iteration. Each iteration contains also information about the results obtained at its end - a table of confusion and other measures, with every example's evaluation.

The second type of measurements is taken at the end. The core feature is the final grammar which contains information about all obtained rules and symbols. The rules are described with identical parameters as during the learning process. It also contains a results summary - a table of confusion and some basic classification statistical measures - sensitivity, specificity, precision, F1, and fitness. The last included feature is a set of examples and their evaluation.

The final step of the simulation is creating the report with the collected data.

B. Report structure

The whole report is divided into seven separate tabs, each containing differently visualized data. Each tab is described by one of two possible kinds of labels. One label, *Learning*, denotes tabs with information about learning the process. Those labeled as *Grammar* describe only the final grammar.

Results evaluation is the first possible tab, which contains the information about the evolution of basic measures (fitness, sensitivity and specificity) during the entire simulation. They are visualized using an interactive chart. It is possible to resize it, exposing the demanded parts of the simulation.

The *Grammar evolution* tab contains the largest amount of significant data. It is the main tool for learning process analysis. It collects the information about the system state in each step. In the top part, the key explanation of the applied symbols is placed. The bottom part is filled with a horizontal, scrollable panel, with separate sections for each iteration. Each iteration is divided into four parts - first, denoted with a "+" sign, contains rules that were added (and remained) in grammar during a given iteration. Each added rule contains the information about the process that led to its creation on its left side, and its possible parents or children. The second section is labeled with "0" sign - in this section rules that retain from the previous iteration occur. They do not have special symbols besides the ones reserved for those which were parent rules for the genetic algorithm. The third section is marked by "-" sign. It contains rules which were removed during iteration. The reason for their removal is on their right side. What is important, a rule that was added or used as a parent in the genetic algorithm can also occur in this section. It means that it was removed at the later part of the iteration. All of the rules in the abovementioned sections show additional information about them while keeping the mouse cursor over them. That information contains a given rule's age, fitness, usages in proper and invalid parsing, profit and debt points. All of the rules in the first section are marked with blue color and are of the same size, however the rules in last two sections differ in color and size. Bigger rules have greater fitness value than the smaller ones. Rules age is also distinguished - rules with brighter green color are younger than the darker rules. Additionally, they are sorted by their fitness. The last section contains test set examples. Positive ones are denoted with tick symbol and the negative ones with the cross symbol. The examples differ with their background color - green means that a particular example was correctly classified in this iteration and the red one means that it was not so.

The *Rules lifespan* tab includes information about rules lifespan by placing bars on a timeline. The bars for each rule span between the rule's creation and its annihilation. Each rule can have multiple bars because it can appear in the grammar many times throughout the simulation.

The *Results* tab includes the results obtained with the final grammar. The first part contains values of basic measures and the second one - the table of confusion.

The *Examples evaluation* tab covers the result of learning set classification performed using the final grammar. The examples are visualized exactly like on the Grammar evolution tab.

The *Final Grammar* tab allows for analyzing the rules which were retained to the final grammar. These rules are represented in the same way as in the Grammar evolution tab.

IV. Example of Usage

As previously mentioned, in the given chapter we would like to present one of exemplary test reports, which resulted in some valuable insight and initiated the modification of covering approach. The test was performed using a context-free language $\{ac^m \mid m \geq 1\} \cup \{bc^m \mid m \geq 1\}$ *over* $\Sigma = \{a, b, c\}$ [8] and standard parameters.

The analysis started with the *Results evaluation* tab. It helped to notice quickly that the collected main performance features repeat in a certain pattern (Fig. 1).

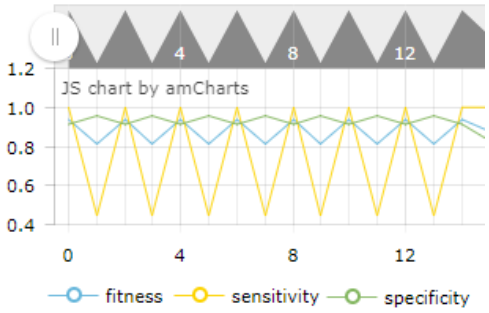


Fig. 1. Main measures evolution over time.

Fitness is presented using blue color, sensitivity and specificity with yellow and green respectively. All of those measures accept values from 0 to 1. Performance depends on the rules contained in the grammar in a given iteration, which were collected in the *Grammar evolution* tab.

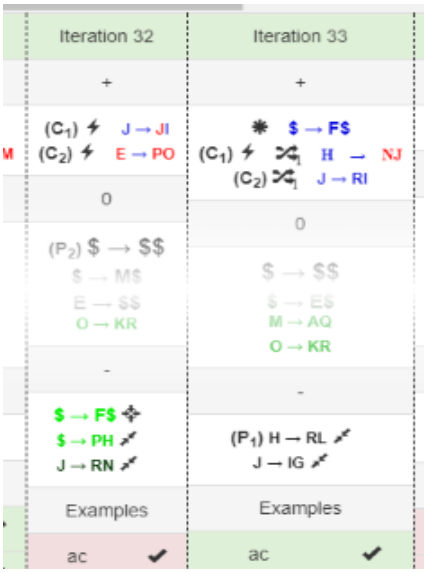


Fig. 2. Grammar rule-set evolution over time.

Fig. 2 represents part of the exemplary tab. It is titled with iteration number. Each part is divided into *added* (+), *remained* (0) and *removed* (-) sections. All the symbols contained in this tab are described in Fig. 3.

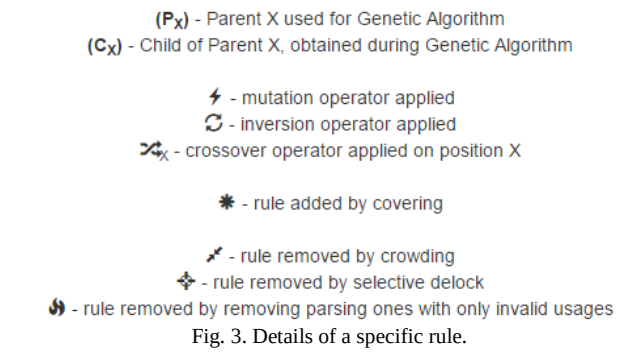


Fig. 3. Details of a specific rule.

The star mark denotes rules added with covering (Fig. 4), the thunder and the crossed arrows application of the genetic algorithm – *mutation* (mutated symbols with red color) and *crossover* (exchanged symbols with red color).

Fig. 4. Rule added with covering.

P_1 and P_2 are parents for the genetic algorithm in this iteration, and C_1, C_2 are their children (Fig. 5).

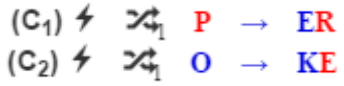


Fig. 5. Example of genetic algorithm children, with mutation and crossover operators applied.

The rules removed using *delock* algorithm are marked with the crosshair symbol. Crowding uses the face-turned arrows as a symbol (Fig. 6).

Fig. 6. Rules removed using crowning technique.

The rules in the *Remained* section are marked with different shades of green, depending on their age (the light green rules are the youngest). The rules also vary in size – those with higher fitness value are bigger. The detailed data about iterations show that in every second iteration one of the rules is removed, and then, in the next step, added again. It is removed because the *delock* algorithm selects to remove only those rules which have invalid usages, and in such case, there is only one considered rule. Each iteration rule details can be easily displayed by placing a mouse cursor over it. It shows information about current rule's features – fitness, proper usages, invalid usages, profit, debt and age (Fig. 7).

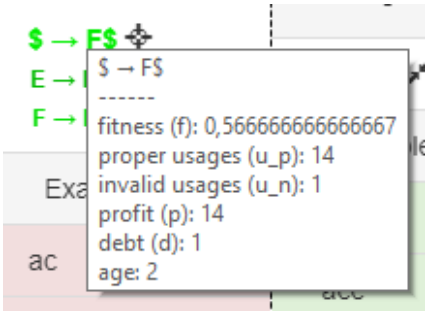


Fig. 7. Details of a specific rule.

However, in the next iteration, GCS tried to parse positive examples that had been first classified as negatives. In order to do that, a covering algorithm was called, and adding the same rule was the only option. This pattern, repeated until a rule added with the genetic algorithm interrupted it, blocked and slowed down the learning process. What is worse, in some other cases it could even make learning impossible. The simulation ended without obtaining maximal performance. The results were presented in the *Results* tab (Fig. 8).

The analysis of this report revealed the need for making the system more aware of learning process history. Adjusting simulation parameters during learning process using collected history features could improve the performance of learning algorithm when a similar problem would occur.

The sample report is available online at *http://lukasz.culer.staff.iiar.pwr.edu.pl/reports/ex1.html*.

Results

fitness: 0,94

sensitivity: 0,89

specificity: 0,96

precision: 0,89

f1: 0,89

Example results summary

		Actual class	
		Positive	Negative
Predicted class	Positive	8	1
	Negative	1	22

Fig. 8. Final induction results.

V. Applications

The presented tool lays a solid foundation for applications in learning process analysis. A few of them are described below:

1) *Comparing algorithm work with different parameters* - visualization of main measures and legible representation of rules for each iteration makes it accessible. Comparing simulations with different parameters allows us to find those, whose desired parameters (like fitness, ability to generalize, learning speed) could be optimized.
2) *Issues with learning stability* – they could be detected with ease, only with basic measure change or a number of added or removed rules observation. It could also be checked statistically when comparing to numerous other simulations.
3) *Adjusting the number of symbols and rules in a simulation* - due to the information about the usages of a given rule in every iteration, it is possible to detect the situations when a number of symbols or rules are not adjusted. For instance, removing a crucial rule while adding a new one for parsing may suggest that the learning algorithm cannot fit into a given rule limits.
4) *Significance of rules in grammar and identity check* – the final grammar and created structures analysis could find those essential for algorithm performance. The rules and structures repeating throughout multiple learning cycles could be detected. Comparing them along many simulations allows us to eliminate the redundant ones and helps to verify their identity.
5) *New operators benchmarking* – the grammar evolution tab creates a template where new operators can be easily added with desired parameters, without creating any new reporting unit. The premade template reduces the time needed to implement and test new operators and features.

VI. Conclusions

The reports generated with the created visualization tool proved to be helpful in making insights essential for improving GCS performance. They allow to compile and analyze data in ways that would be noticeably more problematic in other situations. Substantial data presented in a clear way promises even more future conclusions drawn from developed visualizations.

VII. Future Work

One of currently developed features is the final grammar visualization with the interactive graph that shows the relations between symbols and rules. The start symbol is highlighted with a red triangle symbol in the background. The terminal symbols are marked with blue squares in the background, while non-terminal - with a black ellipse. Terminal rule relation is expressed using the blue line. Non-terminal rules are visualized using the black and red lines. The black line leads to the first symbol on the right side of the rule, and the red one - to the second symbol of rule's right side (Fig. 9).

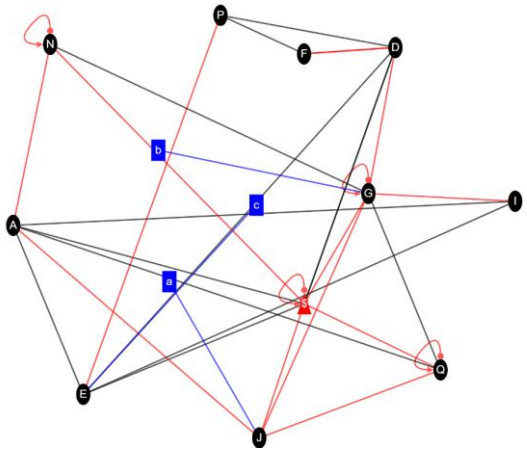


Fig. 9. Final grammar interactive graph – work in progress.

This feature will allow for representing grammar structure in a way which is clear and accessible for analysis. Graphic representation will make identifying structures easier, as well as comparing complexity of two grammars from different learning cycles.

Another extension that is planned to be added in the future is an interactive CYK parsing table for each example in every iteration. The CYK table, instead of presenting only one of parsing trees, will enable to us to analyze the parsing process of certain examples more deeply. It will be simpler to detect structures which are more durable (due to the existence of many equivalent parsing sub-trees) or those more fragile (due to the availability of only one sub-tree, or the fact that it was built using low fitness rules).

Each property (the fitness of rules that build up parsing trees, the number of alternative sub-trees) will be visualized with colors and different types of lines. The new feature will allow us to easily compare approaches to parsing given examples.

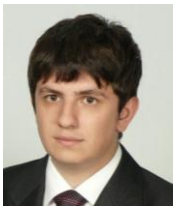
References

[1] C. De la Higuera, *Grammatical Inference: Learning Automata and Grammars*, Cambridge University Press, 2010.
[2] J. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning, an Artificial Intelligence Approach*, R. S. Michalski *et al.*, Eds., Morgan Kaufmann, San Francisco, vol. II, pp. 593–623, 1986.
[3] O. Unold, "Context-free grammar induction with grammar-based classifier system," *Archives of Control Science 15*, pp. 681–690, 2005.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, 1st edition, 1979.
[5] D. Younger, "Recognition and parsing of context-free languages in time n^3," *University of Hawaii Technical Report*, Department of Computer Science, 1967.
[6] O. Unold and L. Cielecki, "Grammar-based Classifier system," in *Issues in Intelligent Systems: Paradigms*, O. Hryniewicz *et al.*, Eds., Warsaw: EXIT Publishing House, 2005, pp. 273-286.
[7] O. Unold, "Playing a toy-grammar with GCS," in *Proc. International Work-Conference on the Interplay Between Natural and Artificial Computation*, Berlin: Springer, 2005, pp. 300-309.
[8] Y. Sakakibara and M. Kondo, "GA-based learning of context-free grammars using tabular representations," *ICML*, pp. 354-360, 1999.

Olgierd Unold is an associate professor in the Department of Computer Engineering of the Wrocław University of Science and Technology. He received an MSc degree in automation systems in 1989, an MSc degree in information science in 1991, and PhD and DSc degrees in computer science in 1994 and 2011, respectively. His research interests focus on adaptive machine learning methods.

Agnieszka Kaczmarek is a Ph.D. student in the Department of Computer Engineering of the Wrocław University of Science and Technology. She received her Master's Degree in 2014. Her main research interests are artificial intelligence and bioinformatics.



Łukasz Culer is a Ph.D. student in the Department of Computer Engineering of the Wrocław University of Science and Technology. He received his Master's Degree in 2014. His scientific interests include machine learning and image processing.